2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

# Learning Local Planners for Human-aware Navigation in Indoor Environments

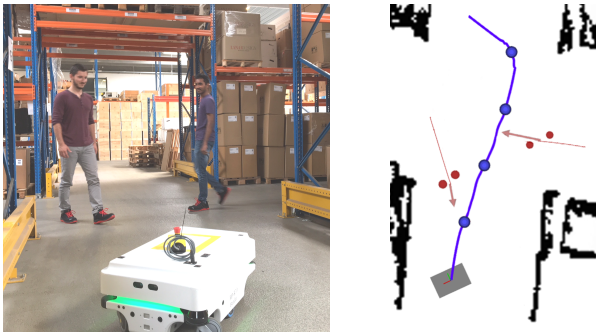
Ronja Güldenring*[1], Michael Görner[2], Norman Hendrich[2], Niels Jul Jacobsen[1], Jianwei Zhang[2]

***Abstract*— Established indoor robot navigation frameworks build on the separation between *global* and *local* planners. Whereas global planners rely on traditional graph search algorithms, local planners are expected to handle driving dynamics and resolve minor conflicts. We present a system to train neural-network policies for such a local planner component, explicitly accounting for humans navigating the space. DRL-agents are trained in randomized virtual 2D environments with simulated human interaction. The trained agents can be deployed as a drop-in replacement for other local planners and significantly improve on traditional implementations. Performance is demonstrated on a MiR-100 transport robot.**

## I. INTRODUCTION

Autonomous indoor navigation of wheeled robots is one of the fundamental problems in robotics [1]–[3]. In this context, the environment is often modeled in 2D-space and grid approximations are used to represent free space and estimated obstacles [4], which enables straightforward collision testing and path planning. Systems employ traditional algorithms, such as *A**, *Dijkstra's algorithm*, or *Probabilistic Roadmaps*, to plan motion trajectories. While this graph-based planning is computationally cheap, the resulting trajectories usually do not adhere to the kinematics of the robot and do not account for dynamic obstacles. These tasks are forwarded to a second planning component, i.e., the *local planner*, which receives current sensor inputs and generates control signals that move the robot along the planned trajectory. In practical application, local planners often employ established parameterized approaches like the *Dynamic Window Approach* [5] or *Elastic Bands* [6], [7]. These approaches can also be configured for nonholonomic systems, e.g., *Ackermann* steering vehicles.

Traditional local planners only include perceived obstacles as momentarily static because the current position of obstacles can be directly measured through depth sensors. However, in populated indoor environments, many perceived obstacles are caused by humans moving about, and navigating as if they would remain static can result in unwanted close encounters. One approach to include environment dynamics in the planner is to fit and rely on data-driven models to recognize other agents, predict their movement intention, and include the resulting trajectories in local optimization [8], [9]. By contrast, we describe a system to train and deploy neural-network policies for a local planner module with established reinforcement-learning paradigms. Agents can be trained in simulation environments based on recorded maps of their actual physical workspace. To achieve good behavior in close encounters with humans, we explicitly integrate reactive human movement based on crowd simulation models.



(a) (b)

Fig. 1: **(a)** The MiR-100 transport robot [10] navigating a warehouse with walking humans. **(b)** The corresponding scene in our DRL-agent simulation and training setup. It includes static obstacles (black), the robot (gray rectangle), the global motion plan with waypoints (blue), and pedestrians (red). Two circles represent the legs of each pedestrian, the arrow shows the walking direction and speed, while the fine line shows the walking history.

## II. RELATED WORK

With the recent successes of deep reinforcement learning approaches, indoor robot navigation has also become a popular target application [11]–[13]. Tai et al. demonstrated a learning setup for a differential-drive robot with a circular footprint that is trained entirely in simulation and navigates to relative goal locations [14]. The agent receives a low-dimensional LIDAR sensor signal and learns to circumnavigate convex static obstacles. Kato et al. [15] improved on this idea by integrating a global topological planner. They proposed to model humans as pairs of circles but simplified the motion patterns to constant linear paths for each circle.

To account for realistic dynamic interaction with pedestrians, Kretzschmar et al. [8] presented a generative model, trainable from data, for human navigation behavior. The system operates through Monte-Carlo-Sampling on Voronoi maps and deployment on a physical system requires a dedicated perception module to detect human obstacles.

Fan et al. [16] extended an agent trained in simulation to support reliable localization in crowded scenes. The agent

*Corresponding author ⟨rog@mir-robots.com⟩
[1] with MiR Mobile Industrial Robots A/S, 5220 Odense SØ, Denmark.
[2] with Department of Informatics, University of Hamburg, Germany.
This research was partially funded by the German Research Foundation (DFG) and the National Science Foundation of China in project Crossmodal Learning, TRR-169.

monitors its localization belief and changes the controller to navigate to nearby landmarks as needed. Of course, a loss of localization usually only occurs in special places, e.g., the canteen.

Faust et al. [17]–[19] presented a system that trains a local navigation module in kinematic simulation, directly optimizing hyperparameters at the cost of severe computational overhead. The resulting policy is applied as a local planner in a global roadmap approach, where it influences roadmap generation. Their physical evaluation demonstrates impressive results but show-cases a differential-drive system with a small round footprint.

A different DRL approach for dynamic multi-agent navigation was proposed in the work of Everett et al. [20]. Instead of working with raw sensor data, a recursive LSTM network is fed with preprocessed agent positions during each time step, learning to focus on the nearest other agents.

In this paper, we provide a full system to simulate and train RL agents for local navigation of indoor robots in the presence of walking humans. Our novel training environment includes simulated reactive human agents based on crowd simulation approaches and directly integrates into the ROS navigation stack [21]. We exemplarily investigate the effect of different state representations for two typical deep network architectures under appropriate shaped rewards. The trained agents outperform baseline local planners, show human-aware driving behavior, and work well in tests on the real robot (Fig. 1).

## III. METHODS

### A. SIMULATION

Despite recent progress in computer-vision-based navigation approaches, 2D laser scanners (LIDARs) remain the most popular sensors for indoor robot navigation. The sensors are affordable and generate high-quality low-noise range measurements. Therefore, we decided to restrict our simulation environment for training the DRL-agent to a 2D world, taking into consideration only 2D laser scan sensors. Compared to 3D simulators (e.g. [22]–[24]), this significantly reduces the computational effort, contributing to faster training. Also, the 2D range scanner simulation approximates the real world sensors sufficiently well to expect an easy transfer from simulation to the real world.

Fig. 2 illustrates the overall architecture of our system, which combines the existing tools PedSim [25] and Flatland [26] to provide a ROS-based [27] environment that generates realistic and natural robot and pedestrian movements.

*PedSim Crowd Simulator:* Realistic simulation of interacting walking humans is not easy and can lead to computationally expensive representations [8]. In PedSim, individual pedestrians move according to Helbing's social forces model [28], [29]. The model combines different potential fields, arising from the goal, static obstacles and other pedestrians. Potential field approaches are simple enough to compute to utilize them in accelerated simulations for RL, but they can also get the agents stuck in local minima, increasing significantly with the complexity of the static map. For that

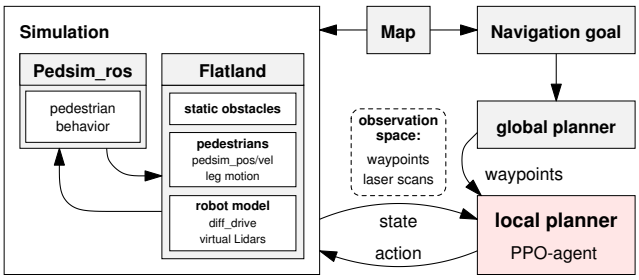
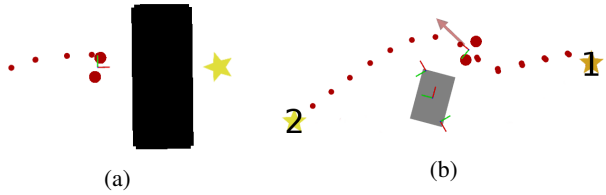



Fig. 2: Main components of our simulation environment. Human walking trajectories are calculated by Pedsim_ros. The Flatland simulator manages obstacles and the moving robot, and also generates pedestrian leg motion patterns. The DRL-agent observes the simulated laser scans and learns to control the robot motion to follow the target waypoints.




(a) (b)

Fig. 3: **(a)** A pedestrian tries to reach the waypoint behind a static obstacle, but is stuck in a local minimum of the potential field. **(b)** The pedestrian walks from waypoint 1 to waypoint 2, but the robot blocks the direct path. The red dotted line shows the path taken by the pedestrian instead.

reason, we disabled the influence of static objects on pedestrians in our simulation and accepted in return pedestrians walking through static objects, resulting in a more unrealistic behavior. Furthermore, we extend the force model with an additional exponential repellent force arising from the robot center. This prevents pedestrians from walking into the robot footprint. But as a crucial difference to the pedestrian forces, the robot force is only applied after the robot stands still for some time $t_{reac}$. This modification simulates pedestrians who expect the robot make way during operation, but actively avoid a stationary robot. Fig. 3 shows example behaviors of a simulated pedestrian.

*Flatland Simulator:* We use Flatland [26] as base 2D simulator. It can run simulations significantly faster than realtime, a static environment can be created simply by loading a previously recorded grid map, and provides interfaces compatible with ROS's move_base stack. Additional objects, composed of geometric primitives, can be loaded and removed dynamically.

The model of the robot consists of a single polygon that represents the footprint of the real robot. To reduce computation time, we do not simulate physical aspects like joint translations, motors torques, friction properties, and acceleration profiles. The laser scan sensors are simulated through the existing laser plugin of Flatland, which also models Gaussian measurement noise. We add two artificial

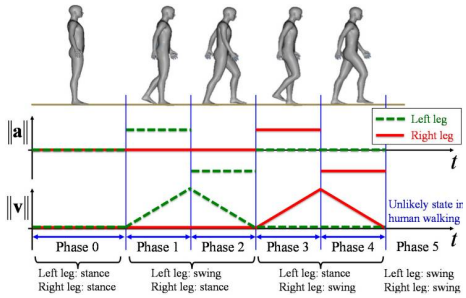



Fig. 4: Different phases of a simple leg movement model for human walking. One leg swings at a time and follows a triangular velocity function, while the other leg remains stationary [30]. Key parameters, such as velocity and physique, are randomized in our application.

laser scans "static_laser" and "ped_laser" to the center of the robot, which perceive only static obstacles or pedestrians, respectively. These auxiliary scans allow distinguishing between both obstacle types during rewarding (see section III-C.5) and are merged to yield the full simulated sensor output. The differential drive of the robot is modeled with the Flatland diff_drive plugin.

*Pedestrian Plugin:* To implement the human trajectories calculated by PedSim in Flatland, we created a pedestrian plugin. In the 2D world, each pedestrian is modeled with two circles, representing the legs as perceived in the robot laser scanner. The plugin takes PedSim's calculated velocity and position of each pedestrian and then adds a leg-swinging motion. One leg at a time swings according to a triangular velocity function. To yield a commanded overall body velocity of $v$, the leg linearly accelerates to a maximum velocity of $4{\cdot}v$, followed by a constant deceleration to $0\,\mathrm{m/s}$. During that period of time, the other leg remains in the same position. As real humans have different walking and movement behaviors, we apply domain randomization to key model parameters of the walking pattern, such as leg velocities, leg size and spacing. Fig. 4 illustrates the walking model pattern.

## B. TASK SETUP

The task of the RL-agent (local planner) is to navigate successfully from a start to a goal position along a given global path while reacting to local objects that have not been considered by the global plan. The agent gets a representation of its environment as input and it interacts with the environment by controlling the translational and rotational velocity $(v, \omega)$ of the robot. We consider two kinds of objects: global static objects (black) and pedestrians (red). Global static objects are fixed objects that are listed in the global map. Pedestrians are moving objects that are not present in the global map, but are detected by the laser scan sensors. For each episode, a new random start and goal position is picked, resulting in a global plan computed by the global planner. Thereafter pedestrians, that either cross the path, walk along the path, or stand around close to the path, are spawned randomly. The walking speed of each pedestrian is sampled from a normal distribution and is on average $0.6\,\mathrm{m/s}$—slower than the robots maximum velocity of $0.8\,\mathrm{m/s}$. An episode is considered a success if the robot reaches the final goal within 0.4 m. Fig. 1b shows an example episode.

## C. AGENT MODELS

*1) Deep Reinforcement Learning Approach:* We use Proximal Policy Optimization (PPO) [31] as our main Reinforcement Learning approach and build on the implementation from the open-source project stable-baselines [32]. The deep learning part is realized in the Tensorflow framework [33] and allows us to define policy-networks applying Adam Optimization [34].

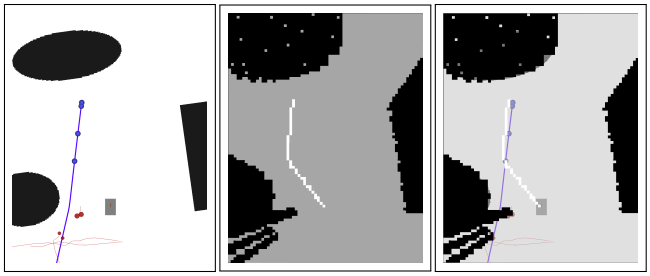
*2) State Representations:* The observation state that has been identified as relevant for the agent to be able to fulfill the task includes:

- Waypoints. The global plan is downsampled to a number of waypoints with a fixed distance to each other. A vector of the next $n_{wp}$ sequential waypoints $[x_i, y_i]$ is provided.
- Laser scan data. In the simulation environment, the static_laser and ped_laser scan data are merged, covering 360° around the robot, and providing a distance value for each angle increment. The vector is discretized to a resolution $res$. While such a laser does not exist on the real robot, we can fuse multiple orthogonal scanners to yield the same representation there.

We compare two different state representations:

*a) RawData Representation:* The laser scan vector of distance values and the $(x, y)$-positions of the $n_{wp}$ waypoints are directly fed into the network.

*b) Image Representation:* Convolutional Neural Networks (CNNs) have been particularly successful with images in the past years. For that reason, we explore an image-based observation space representation, too. Fig. 5 shows a



(a) (b) (c)

Fig. 5: Image representation constructed from laser scan and waypoint vector. **(a)** Example scene with the robot pointing upwards. **(b)** The input image generated from the scene in robot-centric orientation. The image includes a white line from the robot location along the next waypoints, as well as black lines from each scan line pointing away from the robot, starting at the reported distance to an obstacle. White pixels: goal trajectory; gray pixels: free space; black pixels: obstacles or unknown. **(c)** Overlay of (a) and (b).

TABLE I: 5-layered 1D-Conv-network.

| Layer | Type | Activation | Size | Filter Size | Filter Stride |
|---|---|---|---|---|---|
| 1 | Conv | ReLu | 32 Filter | $[5 \times 1]$ | $[2 \times 0]$ |
| 2 | Conv | ReLu | 64 Filter | $[3 \times 1]$ | $[2 \times 0]$ |
| 3 | FC | ReLu | 256 Neurons | - | - |
| 4 | FC | ReLu | 128 Neurons | - | - |
| 5 | FC | Linear | Output Size | - | - |

TABLE II: 5-layered 2D-Conv-network.

| Layer | Type | Activation | Size | Filter Size | Filter Stride |
|---|---|---|---|---|---|
| 1 | Conv | ReLu | 32 Filter | $[8 \times 8]$ | $[4 \times 4]$ |
| 2 | Conv | ReLu | 64 Filter | $[4 \times 4]$ | $[2 \times 2]$ |
| 3 | Conv | ReLu | 64 Filter | $[3 \times 3]$ | $[1 \times 1]$ |
| 4 | FC | ReLu | 512 Neurons | - | - |
| 5 | FC | Linear | Output Size | - | - |

sample scene and the corresponding generated input image. The robot is always positioned at the lower center, pointing upwards, thus including more space in the robot's driving direction. A white line is drawn from the robot location along the upcoming waypoints.

*3) Action Space:* Continuous as well as discrete action space has been investigated. For the continuous action space, limits for the translational velocity $(0, v_{max})$ and angular velocity $(-\omega_{max}, \omega_{max})$ are defined. The discrete action space allows six discrete actions, that are combinations of translational and angular velocities: $(0, 0)$, $(0, -\omega_{max})$, $(0, \omega_{max})$, $(v_{max}, 0)$, $(v_{max}, \omega_{max}/2)$, $(v_{max}, -\omega_{max}/2)$.

*4) Neural Network Structures:* We use two different network architectures which have already proven successful in similar learning tasks. They are applied for both the Actor as well as the Critic Network of the PPO algorithm and are initialized with orthogonal matrix initialization [35].

Table I shows a 1D-Convolutional-Network that is used in combination with the RawData Representation and is inspired by the work of Long and Fan [13]. It consists of 5 layers: two 1D-Convolutional layers, followed by three fully-connected layers. A ReLU activation function is applied to all hidden layers. The laser scan vector of the RawData Representation is processed by layer one of the network, while the $n_{wp}$ closest waypoints are concatenated with the output of layer three and are then forwarded to layer four. The fifth layer finally maps to the output size that varies according to the applied action space type (continuous: 2, discrete: 6).

Table II provides the network architecture for the Image Representation and is the default 2D-Convolutional Network of stable-baselines [32] and inspired by DQN [36]. It consists of 5 layers: three 2D-Convolutional layers, followed by two fully-connected layers. Again, a ReLU activation function is applied to all hidden layers.

*5) Reward Function:* Structured rewards are a key factor to achieve complient behavior and improve learning speed. The following equations describes the general influences, Table III gives the concrete parameter values used in our evaluation. Our reward function considers four summands regarding the final goal $g$, the closest waypoint $wp$, the obstacles $o$ and the velocity of the robot $vel$.

$$R_t = R_t(g) + R_t(wp) + R_t(o) + R_t(vel) \tag{1}$$

The reward $R_t(g)$ contributes with a high positive constant $R_g$ if the position of the robot $p_r$ reaches the goal position $p_g$ within radius $D_g$.

$$R_t(g) = \begin{cases} R_g & \text{if } \|p_r^t - p_g\| < D_g \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The agent at position $p_r^t$ is awarded for getting closer to the next waypoint $p_{wp}^t$ in equation 3, but it is punished for driving away from it in equation 4. The diff()-function in equation 5 determines the difference between the distance of the agent to the next waypoint of the previous time step and its current distance.

$$R_t^1(wp) = \begin{cases} w_1 \cdot \text{diff}(p_r^t, p_{wp}^t) & \text{if diff}(p_r^t, p_{wp}^t) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$R_t^2(wp) = \begin{cases} w_2 \cdot \text{diff}(p_r^t, p_{wp}^t) & \text{if diff}(p_r^t, p_{wp}^t) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$\text{diff}(p_r^t, p_{wp}^t) = \left\|p_r^{t-1} - p_{wp}^{t-1}\right\| - \left\|p_r^t - p_{wp}^t\right\| \tag{5}$$

$$R_t(wp) = R_t^1(wp) + R_t^2(wp) \tag{6}$$

Of course, the agent is punished for colliding with obstacles, distinguishing between static objects (so) and pedestrians (ped). This enables the agent to approach static objects closer while keeping a higher safety distance from moving objects. If the agent collides with a static object $\in SO$, it results in a high negative constant $R_{so}$ (see equation 8). To train the agent to avoid pedestrians with more space, the agent is already punished negatively if it ends up in the circular area of radius $D_{ped}$ around any pedestrian. Since the pedestrians are moving as well, it is not always possible for the agent to keep the demanded distance to all pedestrians, e.g., in crowded or narrow situations. For that reason, the agent does not get punished for being too close to a pedestrian if it stopped moving for some time $t_{reac}$ and the pedestrian caused the close encounter. This motivates the agent to detect upcoming critical situations and wait for the pedestrians to pass. Equation 9 summarizes the rewarding of interactions with pedestrians.

$$R_t(o) = \min(R_t(so), R_t(ped)) \tag{7}$$

$$R_t(so) = \begin{cases} -R_{so} & \text{if collision with object } \in SO \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

$$R_t(ped) = \begin{cases} 0 & \text{if } \forall ped_i\colon \|p_{ped_i}^t - p_r^t\| > D_{ped} \\ & \textbf{or } vel = 0 \text{ for duration } t_{reac} \\ -R_{ped} & \text{otherwise} \end{cases} \tag{9}$$

TABLE III: Experimental reward parameterization.

| Parameter description | Parameter | Value |
|---|---|---|
| distance threshold to pedestrians | $D_{ped}$ | 0.85 |
| distance threshold within goal is reached | $D_g$ | 0.4 |
| constant for reaching the goal | $R_g$ | 10 |
| constant for being to close to a pedestrian | $R_{ped}$ | 7 |
| constant for colliding with static obstacles | $R_{so}$ | 7 |
| constant for standing still | $R^1_{vel}$ | 0.001 |
| constant for turning only | $R^2_{vel}$ | 0.01 |
| reaction time of pedestrians | $t_{reac}$ | 0.8 |
| weight for approaching a waypoint | $w_I 1$ | 4.5 |
| weight for departing from a waypoint | $w_2$ | 5.5 |

Equation 10 shows the definition of $R_t(vel)$ in more detail. To avoid stalling, the agent is punished for not moving. As this, however, encourages irresolute small rotational motions when the way is temporarily blocked, standing completely still is punished less than turning in place $-R^1_{vel} > -R^2_{vel}$.

$$R_t(vel) = \begin{cases} -R^1_{vel} & \text{if } vel_t = 0 \text{ and } \omega_t = 0 \\ -R^2_{vel} & \text{if } vel_t = 0 \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

## IV. EVALUATION

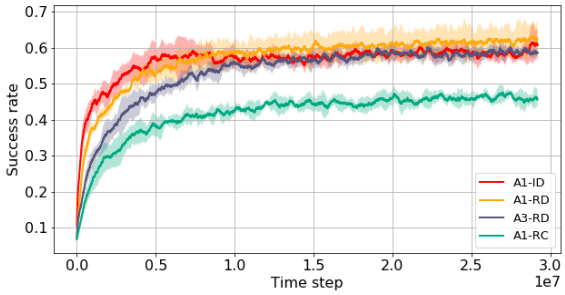
Table IV summarizes the agent setups that have been evaluated and compared with the well-known traditional local planner Dynamic Window Approach (DWA) [5]. Each agent is trained in parallel simulation environments with maps of varying complexity. Agents *A1-ID* and *A1-RD* both apply discrete action space, but differ in their state representation and corresponding deep neural network. *A3-RD* and *A1-RC* are variants of *A1-RD*, where *A3-RD* is trained on a time series by processing a stack of three raw data vectors, where each vector represents a different time stamp with a time offset of 1.5 sec. *A1-RC* applies continuous action space. Each agent setup has been trained and tested five times to estimate performance variances. Since GPU acceleration did not have a significant impact on the overall training time, the agents were trained on CPU only for $\sim 18,5$h.[1]

Fig. 6 shows the training results of all four agent setups, where 6a shows the success rate and 6b shows the time-exceeded rate over time. An episode counts as a success if the agent navigates to the goal without violating a distance threshold to static or dynamic objects (see section III-C.5). The time-exceeded rate sums up all episodes in which the agent did not finish in time, but also did not come to close to any obstacles. For the first one million steps, the image-based representation makes progress slightly faster than agents trained on raw distance information, indicating facilitated training through the explicit spacial representation. However, after this point, all agents with a discrete action space perform comparable, which implies that similar information becomes evident over the course of longer training independent of the processed representation. *A1-ID*, *A1-RD* and *A3-RD* overall show very similar learning curves, while *A1-RD* learns slightly better with a success rate of $\sim 0.62$ and a time-exceeded rate of $\sim 0.3$, resulting in a fairly low collision rate of $\sim 0.08$. The expectations of *A3-RD* are not met since it does not perform significantly better than *A1-RD*, although a time-series of input data is provided. *A1-RC* is significantly less successful, traced back to the higher complexity of controlling in continuous action space.

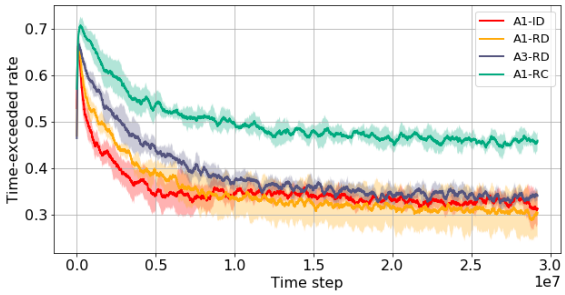
[1] *A1-ID* trained in $\sim$ 35h mostly due to a costly implementation of the image processing

TABLE IV: Agent setups evaluated in section IV.

| Agent Name | A1-ID | A1-RD | A3-RD | A1-RC |
|---|---|---|---|---|
| Action Space | discrete | discrete | discrete | continuous |
| Input Representation | Image | 1-RawData | 3-RawData | 1-RawData |
| Network Architecture | Table II | Table I | | |




(a) Success rate




(b) Time-exceeded rate

Fig. 6: Training results of the different agent setups from Table IV. For each setup, five agents are trained and the average success and time-exceeded rates are determined. The average over 2000 consecutive episodes is plotted.

Fig. 7 shows the test results for all agents as well as the traditional local planner DWA. We used new, unknown maps and tested them in three different testing task setups:

A In a map of average complexity (Fig. 8a) with mainly wide spaces, episodes are spawned according to section III-B.

B Episodes are spawned according to section III-B in a map of medium complexity (Fig. 8b) with narrow corridors.

C Paths without pedestrians are spawned in map 8b. The robot simply needs to follow the globally planned path (aka "pure pursuit").

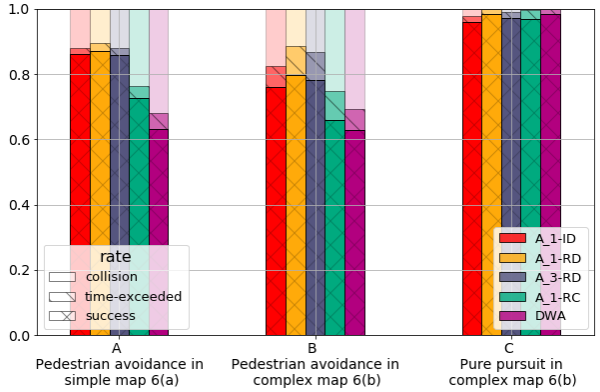



Fig. 7: Test results of the four different RL-agent setups as well as the traditional local planner DWA. All approaches solved 700 dynamic tasks in three different testing task setups each. For each task setup and agent the average success, time-exceeded and collision rate is shown. *A1-RD* and *A3-RD* significantly outperform DWA.

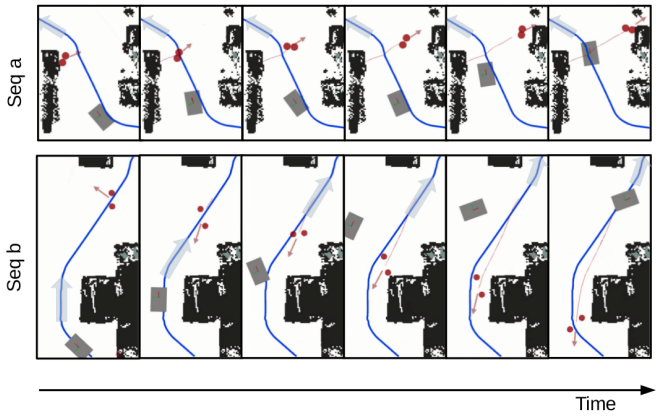
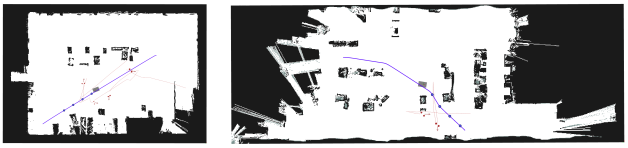



Fig. 9: Example robot behavior. **(a)**: The robot encounters a pedestrian in a narrow passage. It stops and waits until the pedestrian has passed. **(b)**: The robot avoids the approaching pedestrians actively with enough space, before continuing on the global path.

The success rate during testing differs slightly from the one during training, because an episode counts as a success if the robot drives to the goal without collisions. No differentiation between static and dynamic objects is made during testing. All agents, as well as the DWA local planner, are able to follow a path in a static environment (task setup C) with a success rate of 0.97 or better. In task setup A, all of *A1-ID*, *A1-RD* and *A3-RD* reach an average success rate over 0.86, while *A1-RC* only reaches an average success rate of 0.73. For all agents, the time-exceeded rate is relatively low with $\sim$0.03. In task setup B, the time-exceeded rate increases, because complex episodes in corridors occur more frequently. In those situations, the robot stops and waits for the pedestrians to pass, sometimes resulting in deadlocks. Additionally, the collision rate (avoiding immediate collisions) increases from task setup A to B. The comparison between the RL-agents and the DWA local planner should be considered with caution because there are relevant differences between the planners. The DWA planner is allowed to trigger a re-plan from scratch when stuck, while that mechanism was not available for our RL-agents. In contrast, the RL-agents can act on shapes and changes in the environment, while the DWA-planner certainly has no pre-knowledge about the environment. Fig. 9 shows two qualitative representative episodes of how the *A1-RD* trained agent behaves.



(a) (b)

Fig. 8: Test Maps. **(a)** A test map of an average complexity with mainly wide spaces. **(b)** A test map of high complexity with narrow corridors and outer corners.

## V. DEPLOYMENT

As the agents only receive laser scan input, the straight transfer to the real system does not require additional modifications. We deployed the trained agents to the real MiR-100 robot [10], and tested them in situations comparable to the training and test setups (Fig. 1 and Fig. 10). The environment was similar to the maps in Fig. 8, pedestrians walked slowly. The agent learned very well to stop and wait for a pedestrian to pass and to continue driving afterwards. It can occur, that the agent does not avoid obstacles in time and ends up in a stopping-and-waiting state, although there was initially enough space for proper maneuvering. Unfortunately, the robot running the (discrete action) agents often followed the path in an oscillating manner, due to the discretized high yaw-rotation velocities. As a consequence, the robot sometimes approaches walls in narrow corridors closer than in simulation, possibly resulting in a stopping-and-waiting state. We expect that modeling the dynamics properties of the robot controller in simulation can improve this behavior.

## VI. CONCLUSION

We created a system for training an agent to control a mobile robot in environments with active pedestrians. The project is publically available at: https://github.com/RGring/drl_local_planner_ros_stable_baselines and can be used to train and deploy agents on custom maps.

Initial results are very promising, but further application-specific refinements are necessary to allow safe deployment in the real world. Most importantly, the RL-agent should be able to trigger global re-planning or other recovery behaviors in unsolvable situations to avoid deadlocks. Also, the relevant physical properties of the robot should be simulated; this becomes more relevant at higher robot velocities. The results of *A1-RD* and *A3-RD* indicate that the RL-Agent trained on sequences of inputs does not sufficiently account for

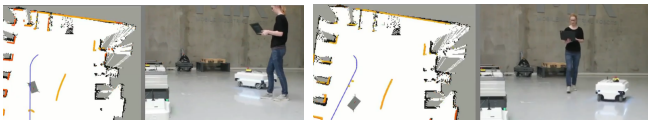


Fig. 10: Real-world test using the MiR-100 robot. See attached video for more examples and scenarios.

the observable motion pattern of the pedestrians yet and might profit from recurrent policy representations. Finally, more advanced pedestrian behaviors, like group walking and gathering, could be modeled, drawing on studies in human movement behavior.

## References


[1] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: A survey," *Robotica*, vol. 33, no. 3, pp. 463–497, 2015. DOI: 10.1017/S0263574714000289.

[2] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1135–1145, Apr. 2016. DOI: 10.1109/TITS.2015.2498841.

[3] T. Kruse, A. K. Pandey, R. Alami, and A. Kirsch, "Human-aware robot navigation: A survey," *Robotics and Autonomous Systems*, vol. 61, no. 12, pp. 1726–1743, 2013. DOI: 10.1016/j.robot.2013.05.007.

[4] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989. DOI: 10.1109/2.30720.

[5] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997. DOI: 10.1109/100.580977.

[6] S. Quinlan and O. Khatib, "Elastic bands: Connecting path planning and control," in *Proceedings IEEE International Conference on Robotics and Automation*, 1993, 802–807 vol.2. DOI: 10.1109/ROBOT.1993.291936.

[7] C. Roesmann, W. Feiten, T. Woesch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," in *ROBOTIK 2012; 7th German Conference on Robotics*, 2012, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/6309484.

[8] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, "Socially Compliant Mobile Robot Navigation via Inverse Reinforcement Learning," *The International Journal of Robotics Research*, vol. 35-11, pp. 1289–1307, 2016. DOI: 10.1177/0278364915619772.

[9] H. Chen, X. Wang, and J. Wang, "A trajectory planning method considering intention-aware uncertainty for autonomous vehicles," in *2018 Chinese Automation Congress (CAC)*, 2018, pp. 1460–1465.

[10] Mobile Industrial Robots A/S. (2019). MiR100 robot, [Online]. Available: https://www.mobile-industrial-robots.com/en/products/mir100/.

[11] N. V. Dinh, N. H. Viet, L. A. Nguyen, H. T. Dinh, N. T. Hiep, P. T. Dung, T. Ngo, and X. Truong, "An extended navigation framework for autonomous mobile robot in dynamic environments using reinforcement learning algorithm," in *2017 International Conference on System Science and Engineering (ICSSE)*, Jul. 2017, pp. 336–339. DOI: 10.1109/ICSSE.2017.8030892.

[12] S. Han, H. Choi, P. Benz, and J. Loaiciga, "Sensor-Based Mobile Robot Navigation via Deep Reinforcement Learning," in *2018 IEEE International Conference on Big Data and Smart Computing*, Jan. 2018, pp. 147–154. DOI: 10.1109/BigComp.2018.00030.

[13] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning," *Computing Research Repository (CoRR)*, 2017. [Online]. Available: http://arxiv.org/abs/1709.10082.

[14] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation," *International Conference on Intelligent Robots and Systems (IROS)*, pp. 31–36, 2017. DOI: 10.1109/IROS.2017.8202134.

[15] Y. Kato, K. Kamiyama, and K. Morioka, "Autonomous robot navigation system with learning based on deep Q-network and topological maps," in *IEEE/SICE International Symposium on System Integration, SII 2017, Taipei, Taiwan*, 2017, pp. 1040–1046. DOI: 10.1109/SII.2017.8279360.

[16] T. Fan, X. Cheng, J. Pan, P. Long, W. Liu, R. Yang, and D. Manocha, "Getting Robots Unfrozen and Unlost in Dense Pedestrian Crowds," *Computing Research Repository (CoRR)*, 2018. [Online]. Available: http://arxiv.org/abs/1810.00352.

[17] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia, "PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-based Planning," in *International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, 2018, pp. 5113–5120. DOI: 10.1109/ICRA.2018.8461096.

[18] H. L. Chiang, A. Faust, M. Fiser, and A. Francis, "Learning Navigation Behaviors End-to-End With AutoRL," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 2007–2014, 2019. DOI: 10.1109/LRA.2019.2899918.

[19] A. Francis, A. Faust, H. L. Chiang, J. Hsu, J. C. Kew, M. Fiser, and T. E. Lee, "Long-range indoor navigation with PRM-RL," *CoRR*, 2019. [Online]. Available: http://arxiv.org/abs/1902.09458.

[20] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *2018 IEEE/RSJ*

*International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 3052–3059. [Online]. Available: https://arxiv.org/abs/1805.01956.

[21] E. Marder-Eppstein *et al.* (2020). ROS Navigation stack, [Online]. Available: https://github.com/ros-planning/navigation.

[22] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, 2004, pp. 2149–2154. DOI: 10.1109/iros.2004.1389727.

[23] Coppelia Robotics. (2019). V-rep virtual robot experimentation platform, [Online]. Available: http://www.coppeliarobotics.com/.

[24] Cyberbotics Inc. (2019). Webots robot simulator, [Online]. Available: https://www.cyberbotics.com/.

[25] B. Okal, T. Linder, D. Vasquez, and L. P. Sven Wehner Omar Islas, *Pedsim_ros*, 2018. [Online]. Available: https://github.com/srl-freiburg/pedsim_ros.

[26] Avidbots, *Flatland*, 2018. [Online]. Available: https://github.com/avidbots/flatland.

[27] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*.

[28] Christian Gloor, *PedSim Crowd Simulator*, 2016. [Online]. Available: http://pedsim.silmaril.org/download/.

[29] P. M. Dirk Helbing, "Social force model for pedestrian dynamics," *Physical Review E 51*, pp. 4282–4286, 1995. DOI: 10.1103/PhysRevE.51.4282.

[30] A. Yorozu, T. Moriguchi, and M. Takahashi, "Improved Leg Tracking Considering Gait Phase and Spline-Based Interpolation during Turning Motion in Walk Tests," *Sensors*, vol. 15, no. 9, pp. 22 451–22 472, 2015. DOI: 10.3390/s150922451.

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *Computing Research Repository (CoRR)*, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347.

[32] A. Hill, A. Raffin, M. Ernestus, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, *Stable Baselines*, https://github.com/hill-a/stable-baselines, 2018.

[33] Martín Abadi, Ashish Agarwal, *et al.*, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, Software available from tensorflow.org, 2015. [Online]. Available: http://tensorflow.org/.

[34] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, Dec. 2014. [Online]. Available: http://arxiv.org/abs/1412.6980.

[35] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *CoRR*, 2013. [Online]. Available: http://arxiv.org/abs/1312.6120.

[36] V. Mnih, K. Kavukcuoglu, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236.